Self-Aware OS and Runtime in Exascale Systems

Roberto Gioiosa*, Adolfy Hoisie*, Darren J. Kerbyson*, Daniel G. Chavarría*,
Abhinav Vishnu*, Sriram Krishnamoorthy*

*Pacific Northwest National Lab
Richland, WA 99352
Email:{roberto.gioiosa, adolfy.hoisie, darren.kerbyson, daniel.chavarria, abhinav.vishnu, sriram}@pnnl.gov

Introduction

Future exascale systems will be composed by millions of heterogeneous components that will be dynamically activated and deactivated as result of contingent conditions, such as opportunities for power/energy efficiency, poor reliability, task/data movement and/or re-allocation of hardware resources. In this context, understanding whether an applications is executing at the maximum expected speed and efficiently performing computation is a hard task. Moreover, each individual component may operate in different states (DVFS, number of active cores/hardware threads, availability of memory), which further increases the complexity of efficiently managing system's resources and may lead to sub-optimal performance or even performance degradation if not well guided.

The programmer could theoretically provide low-level information (when and which core to power off, where to allocate a particular data structure) to guide the system's actions but this would considerably increase the complexity of writing exascale applications. More importantly, some of the information, such as the allocation of global data structures, are not directly under the control of the programmer but are determined by the runtime system [1]. Finally, this effort may have to be repeated for each application/kernel development and whenever external conditions (new system configuration, different input set) change the characteristics of the application.

A self-aware, automated system software (operating and runtime system) has the potentiality of extracting the necessary information from the application execution and the current system conditions. Solving the problem at OS/RT level has a direct impact on programmers' productivity: first the programmer need not have to deal with the system complexity of scheduling a large number of tasks, dealing with faults and efficiently managing hardware resources. Second, an automated system can analyze different parameters, some of which are only known at run time and depends on the particular system and execution environment. As we expect the number of this parameter be large, a manual approach would not be feasible and would probably result in sub-optimal performance and efficiency.

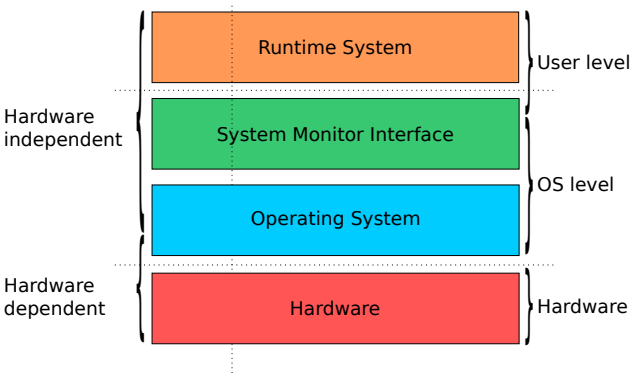

Fig. 1. System software design

However, as we move from the application to lower levels of the software stack we lose behavioral knowledge that can be put to use in optimizing performance and power/energy consumption without unwanted impacts on performance. The runtime system can benefit from utilizing quantitative and predictive application models [2]–[6] to make intelligent decisions, for instance when considering data movement, data replication and computation migration, and will utilize tools provided by the hardware to route power to where it can be most effectively utilized through throttling and power gating.

Figure 1 shows the main components involved in the design of a self-aware automated system software stack. Here follows a brief description of each of them:

a) Runtime system (RT): Runtime software employed to manage execution at exascale must be equipped with tools (i.e., models) that are able to guide dynamic decision-making policies concerning when data migrations are beneficial and weigh the cost of movement against possible alternatives.

b) System Monitor Interface (SMI): The SMI decouples OS and the RT with the main goal of increasing portability. Together with the OS, SMI hides the low-level architecture details enabling the porting of RTs on different machines that possibly run different operating systems. Current OS do not export the level of runtime information that required by dynamic models to make the correct decisions. For example, Unix-base operating systems export system information (`/sys`) and process information (`/proc`) but do not provide information on the level of achieved performance (percentage of peak performance), efficiency (power/energy consumption per computation task) or the quality of the results (number of faults, correctable/uncorrectable errors).

c) Operating System (OS): Current OS used in HPC are usually disconnected form the RT systems and may take sub-optimal decision. For example, memory is allocated according to a flat model, with the only exception being NUMA domains (`numactl`), without considering where computation is assigned or where other global data structures are allocated. Only minimal support is provided for scheduling tasks on cores/hardware threads that share internal resources. Finally, resources such as accelerators or memory on-device are considered second-class resources and are not directly managed. OS designed for exascale system should provide an higher level of flexibility and tighten the integration with RT. This can be accomplished by 1) providing (through SMI) the required information with low-overhead, and 2) allowing RT to specify extra *QoS* parameters when allocating or managing resources (on which memory device a global data structure should be allocated, whether the memory should be contiguous, the desired level of reliability or power consumption of a computation task).

d) Hardware (HW): We expect exascale architectures to be more flexible than current ones. In particular, we expect future hardware components will allow fast transition among low-power states and the ability to dynamically allocate some of the hardware resources on demand. More importantly, we expect those performance and power/energy capabilities be exposed to the system software (software-controlled), similar to IBM POWER hardware thread priority [7].

Challenges Addressed

Self-aware OS/RT systems have the ability to adapt to the current situation and react to runtime events. The increased OS flexibility will provide the RT with mechanisms to efficiently specify how to allocate hardware resources. Both these characteristics have direct impact on the most important exascale challenges, such as performance, power/energy, locality and reliability.

Maturity

There has been previous work on dynamic allocation of resources according to application's characteristics [5], [6], [8], in field of performance modeling of parallel applications on large machines [4], [9], [10] and efficient communication runtime systems [11]–[13]. These systems, however, lack the OS/RT integration that is necessary to meet the efficiency levels required by exascale systems.

Uniqueness and Novelty

Exascale systems pose new challenges that are unique to the exascale era (heterogeneity, large number of components, power constraints) and that make previous solutions unable to meet exascale requirements. In particular, current OSs/RTs do not provide the required flexibility and adaptiveness and novel models and modeling techniques should be developed. Proposed solutions [4]–[6], [13] address the problem at OS, RT or architecture level. While this has been proved enough for petascale systems, exascale constraints (40 GFLOPS/Watt) require a much tighten integration of all system's components.

Applicability

An automatic self-aware system will allow users to explore possible trade-offs among performance, power and reliability and alternative design choices that would be to expensive to explore manually. The output of these exploration studies can be used as feedback to other areas, including programming models, computer architectures and algorithms.

Effort

The development of a self-aware OS/Runtime systems is fundamental for the development of other system components, such as the programming model runtime. As such, the effort should be developed in the early part of the exascale roadmap. Given the complexity of tackling the whole system software stack and the variety of design choices, several coordinated approaches can be performed in parallel in order to achieve prototypes development in the early part of the roadmap.

REFERENCES

[1] J. Nieplocha, B. Palmer, V. Tipparaju, M. Krishnan, H. Trease, and E. Aprà, "Advances, applications and performance of the global arrays shared memory programming toolkit," *Int. J. High Perform. Comput. Appl.*, vol. 20, no. 2, pp. 203–231, May 2006. [Online]. Available: http://dx.doi.org/10.1177/1094342006064503

[2] D. Kerbyson, A. Vishnu, K. Barker, and A. Hoisie, "Codesign challenges for exascale systems: Performance, power, and reliability," *Computer*, pp. 37–43, 2011.

[3] D. Kerbyson, A. Vishnu, and K. Barker, "Energy templates: Exploiting application information to save energy," in *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*. IEEE, 2011, pp. 225–233.

[4] F. Petrini, D. J. Kerbyson, and S. Pakin, "The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of asci q," in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, ser. SC '03. New York, NY, USA: ACM, 2003, pp. 55–. [Online]. Available: http://doi.acm.org/10.1145/1048935.1050204

[5] C. Boneti, R. Gioiosa, F. Cazorla, and M. Valero, "A dynamic scheduler for balancing HPC applications," in *SC '08: Proc. of the 2008 ACM/IEEE conference on Supercomputing*, 2008.

[6] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: making dvs practical for complex hpc applications," in *Proceedings of the 23rd international conference on Supercomputing*, ser. ICS '09. New York, NY, USA: ACM, 2009, pp. 460–469. [Online]. Available: http://doi.acm.org/10.1145/1542275.1542340

[7] B. Sinharoy, R. Kalla, W. J. Starke, H. Q. Le, R. Cargnoni, J. A. Van Norstrand, B. J. Ronchetti, J. Stuecheli, J. Leenstra, G. L. Guthrie, D. Q. Nguyen, B. Blaner, C. F. Marino, E. Retter, and P. Williams, "Ibm power7 multicore server processor," *IBM J. Res. Dev.*, vol. 55, no. 3, pp. 191–219, May 2011. [Online]. Available: http://dx.doi.org/10.1147/JRD.2011.2127330

[8] C. Boneti, R. Gioiosa, F. Cazorla, J. Corbalan, J. Labarta, and M. Valero, "Balancing HPC applications through smart allocation of resources in MT processors," in *to appear in the 22nd IEEE Int. Parallel and Distributed Processing Symp. (IPDPS08)*, Miami, FL, 2008.

[9] A. Hoisie, G. Johnson, D. J. Kerbyson, M. Lang, and S. Pakin, "A performance comparison through benchmarking and modeling of three leading supercomputers: blue gene/l, red storm, and purple," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1188455.1188534

[10] K. Davis, K. Barker, and D. J. Kerbyson, "Performance Prediction via Modeling: A Case Study of the ORNL Cray XT4 Upgrade," *Parallel Processing Letters*, vol. 19, no. 4, December 2009, pPL link.

[11] J. Nieplocha, B. Palmer, V. Tipparaju, M. Krishnan, H. Trease, and E. Aprà, "Advances, applications and performance of the global arrays shared memory programming toolkit," *Int. J. High Perform. Comput. Appl.*, vol. 20, no. 2, pp. 203–231, May 2006. [Online]. Available: http://dx.doi.org/10.1177/1094342006064503

[12] A. Vishnu, S. Song, A. Marquez, K. Barker, D. Kerbyson, K. Cameron, and P. Balaji, "Designing energy efficient communication runtime systems: a view from pgas models," *The Journal of Supercomputing*, pp. 1–19, 2011. [Online]. Available: http://dx.doi.org/10.1007/s11227-011-0699-9

[13] ——, "Designing energy efficient communication runtime systems for data centric programming models," in *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, ser. GREENCOM-CPSCOM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 229–236. [Online]. Available: http://dx.doi.org/10.1109/GreenCom-CPSCom.2010.133